Diskreter Logarithmus

Josef Schmeißer und Fabian Grotz

6. Mai 2016

Inhaltsverzeichnis

1 Definitionen [5] {Fabian Grotz}

In Kryptosystemen wird eine Funktionalität gefordert die es ermöglicht, geheime Daten auszutauschen. Dies wird durch den Einsatz von Symmetrischen Übertragungsverfahren bewerkstelligt, bei dem beide Parteien einen gemeinsamen geheimen Schlüssel besitzen. Um diesen geheimen Schlüssel auszutauschen ohne das ein Angreifer darauf zugriff hat, verwendet man Algorithmen und Methoden die sehr oft auf dem Problem des Diskreten Logarithmus aufbauen.
Der Diskrete Logarithmus ist sehr schwer zu berechnen. Die erzeugende Exponentialfunktion hingegen ist sehr einfach im Sinne der Komplexität zu berechnen was den Diskreten Logarithmus zu einem perfekten Kandidaten für einen erstmaligen Schlüsselaustausch macht.
Zunächst einmal beschäftigen wir uns deswegen mit den mathematischen Grundlagen die für eine Betrachtung des Diskreten Logarithmus-Problems unabdingbar sind.

1.1 Gruppen

Es sei $(G, \cdot)$ eine Gruppe wie wir sie bereits aus anderen Vorlesungen kennen.
Die Ordnung der Gruppe entspricht der Anzahl der Elemente in G.
Die Ordnung eines Elements $\alpha \in G$ ist die Zahl n, für die gilt: $\alpha^n = e$, wobei e das neutrale Element bzgl. $\cdot$ ist.
Es sei $n \in N$ und $\alpha^n = 1$ sowie $\alpha^{n/p} \neq 1$ für alle Primteiler p von n. Dann ist n die Ordnung von α.
Die Gruppe heißt zyklisch, wenn ein $\alpha \in G$ existiert, so dass gilt:

$$\forall a \in G \; \exists i \in N : \alpha^i = \alpha$$

α heißt Generator der zyklischen Gruppe.
Sei $(G, \cdot\, p)$ eine Gruppe und $\alpha \in G$.
α habe eine endliche Ordnung. Man schreibt $\langle\alpha\rangle$ für die von α erzeugte Untergruppe.

1.2 Primwurzel

Es sei $(G, \cdot\, p)$ eine Gruppe, $\alpha \in G$ und $p = |G|$. α heißt Primitivwurzel, wenn α die Ordnung p hat und $ggT(\alpha, p) = 1$.
Es sei $(Z_p, \cdot\, p)$ eine prime Restklassengruppe und $\alpha \in Z_p$.
Weiterhin seien $p_1, p_2, \cdots, p_n$ die Primfaktoren von G.
α ist eine Primitivwurzel, wenn für $1 \leq r \leq n$ gilt $\alpha^{pr} \neq 1$.

1.3 Beispiel

Als Beispiel dient die Primzahl p = 13 und die dazu gehörige prime Restklassengruppe $G = (Z/13Z)^\times = \{1, 2, \ldots, 12\}$. Zur Primitivwurzel $g = 2$ wird nun die Wertetabelle der diskreten Exponentiation bestimmt.

$$
\begin{aligned}
2^0 &= 1 \equiv 1 \bmod 13 \\
2^1 &= 2 \equiv 2 \bmod 13 \\
2^2 &= 4 \equiv 4 \bmod 13 \\
2^3 &= 8 \equiv 8 \bmod 13 \\
2^4 &= 16 \equiv 3 \bmod 13 \\
2^5 &= 32 \equiv 6 \bmod 13 \\
2^6 &= 64 \equiv 12 \bmod 13 \\
2^7 &= 128 \equiv 11 \bmod 13 \\
2^8 &= 256 \equiv 9 \bmod 13 \\
2^9 &= 512 \equiv 5 \bmod 13 \\
2^{10} &= 1024 \equiv 10 \bmod 13 \\
2^{11} &= 2048 \equiv 7 \bmod 13
\end{aligned}
$$

a	1	2	3	4	5	6	7	8	9	10	11	12
$2^a \equiv mod 11$	1	2	4	8	3	6	12	11	9	5	10	7

Wie man sehen kann sind die Potenzen paarweise disjunkt und die gesamte prime Restklassengruppe kann mithilfe den Potenzen von g erstellt werden. Durch vertauschen und sortieren der Tabelle bekommt man die Wertetabelle für den diskreten Logarithmus.

x	1	2	3	4	5	6	7	8	9	10	11	12
$\log_2$ x	1	2	5	3	10	6	12	4	9	11	8	7

1.4 Diskreter Logarithmus Problem

Voraussetzungen:
Sei $(G, \cdot\, p)$ eine multiplikative Gruppe, $\alpha \in G$ ein Element der Ordnung n und $\beta \in \langle\alpha\rangle$

Problem:
Man berechnet ein a im Bereich $0 \leq a \leq n-1$, so dass $\alpha^a = \beta$ ist. a nennt man den diskreten Logarithmus von β zur Basis α.
Dieses einfach anzumutende Problem ist einer der großen Stützpfeiler, auf die sich unsere Kryptosysteme und Sicherheitssysteme stützen. Die Lösung dieses Problems bedarf im Vergleich zur Diskreten Exponentiation erheblich mehr Rechenaufwand. Die effiziente Lösung des diskreten Logarithmus ist eine Herausforderung der heutigen Mathematik und Kryptologie die uns vermutlich noch eine sehr lange Zeit beschäftigen wird.

1.5 Kleiner fermatscher Satz

Der kleine fermatsche Satz sagt aus, dass bei einer ganzen Zahl a und einer Primzahl p gilt:

$$a^p \equiv a \bmod p$$

Falls a kein Vielfaches von p ist, kann man die Gleichung umformen:

$$a^{p-1} \equiv 1 \bmod p$$

1.5.1 Beweis des kleinen fermatschen Satzes

Wir wollen zeigen, dass für eine Primzahl p und eine beliebige ganze Zahl a gilt: $a^p \equiv a \bmod p$. Wenn man es umformuliert kann man hiermit auch sagen, dass $a^p - a$ durch p teilbar ist.

Ist a durch p teilbar, so gilt bereits $a \equiv 0 \equiv a^p \bmod p$.
Wir beweisen den kleinen fermatschen Satz durch Induktion.

Induktionsanfang: $0^p - 0 = 0$ und das ist durch p restlos teilbar.
Induktionsschritt: Die Behauptung sei wahr für ein bestimmtes a. Somit gilt für a+1:

$$(a+1)^p - (a+1) = a^p + \tbinom{p}{1} a^{p-1} + \cdots + \tbinom{p}{p-1} a + 1 - (a+1)$$

Wobei bei jedem Binominialkoeffizienten gilt, dass

$$\tbinom{p}{k} = \frac{p \cdot (p-1) \cdots (p-k+1)}{1 \cdot 2 \cdots k}$$

und damit auch dass p mit $1 \leq k \leq p-1$ nur im Zähler auftaucht. Da wir angenommen haben, dass p prim ist, tauchen auch sonst im Nenner keine weiteren Teiler auf. Die Binominialkoeffizienten sind demnach also alle durch p teilbar, da mindestens ein p im Zähler ist. Daraus folgt schließlich:

$$(a+1)^p - (a+1) \equiv a^p + 1 - (a+1) = a^p - a \mod p$$

und nach der Induktionsvoraussetzung ist dies durch p teilbar.

Zur Berechnung des Diskreten Logarithmus gibt es noch keine schnellen Algorithmen. Die diskrete Exponentialfunktion hingegen lässt sich in kürzester Zeit sehr einfach berechnen. Dies macht den diskreten Logarithmus zu einer perfekten Einwegfunktion die heute sehr weit in verschiedensten Kryptosystemen im Einsatz ist.
Beispiele hierfür sind das ElGamal-Verschlüsselungsverfahren und das Massey-Omura-Schema, welchen wir uns noch zuwenden werden. Andere Beispiele auf die wir nicht eingehen sind der Diffie-Hellmann-Schlüsselaustausch und der Digitale Signatur Algorithmus.

2 Algorithmen zur Bestimmung des diskreten Logarithmus

Offensichtlich kann der diskrete Logarithmus als Umkehrfunktion der diskreten Exponentialfunktion nicht direkt bestimmt werden. Ein naiver Ansatz, um den diskreten Logarithmus zu einem gegeben Element einer Gruppe zu bestimmen, wäre es alle Potenzen eines gegeben Erzeugers der zyklischen Gruppe der Reihe nach zu bestimmen und das Ergebnis zu vergleichen, bis die entsprechende Potenz gefunden wurde.

Dieser Ansatz ist allerdings nur bei sehr kleinen zyklischen Gruppen zielführend, da die Laufzeit direkt von der Ordnung der Gruppe bestimmt wird. Für eine gegebene Gruppe $\mathbb{G}$ ergibt sich also ein Laufzeit in $\mathcal{O}(ord(\mathbb{G}))$.

Aus diesem Grund soll im Nachfolgenden eine Auswahl an Algorithmen vorgestellt werden, welche das gegebene Problem, den diskreten Logarithmus zu bestimmen, lösen und eine bessere Laufzeitkomplexität besitzen.

2.1 Babystep-Giantstep-Algorithmus {Josef Schmeißer}

Der Babystep-Giantstep-Algorithmus dient der Berechnung des diskreten Logarithmus einzelner Elemente zyklischer Gruppen. Der Algorithmus ist dem einfachen Ausprobieren überlegen, die Laufzeit wird aber, wie wir sehen werden, ebenfalls von der Ordnung der Gruppe bestimmt. Erfunden wurde dieser Algorithmus von Daniel Shanks.

2.1.1 Theorie

Gegeben sei eine zyklische Gruppe $\mathbb{G}$ der Ordnung n, ein Generator g und ein Element der Gruppe, welches wir mit α bezeichnen. Gesucht ist nun ein x, sodass $g^x = \alpha$ gilt. Diese Gleichung kann nun folgendermaßen umgeformt werden: Wir schreiben x als $x = im + j$. Nun bietet es sich an, ein m zu wählen, aus dem ähnlich große Zahlenbereiche für i und j resultieren. Aus der vorhergehenden Gleichung folgt, dass dies für $m := \lceil \sqrt{n} \rceil$ der Fall ist. Schließlich gilt $0 \leq i < m$ sowie $0 \leq j < m$. Nun kann folgende Umformung durchgeführt werden:

$$g^{im+j}\alpha \Leftrightarrow g^j = \alpha(g^{-m})^i \tag{1}$$

Der Algorithmus betrachtet nun beide Seiten dieser Gleichung getrennt und teilt sich somit in zwei Phasen.

In der ersten Phase wird für alle j der Ausdruck g^j berechnet und die jeweiligen Paare (j, g^j) werden in einer Tabelle gespeichert. Aufgrund der kleinen inkrementellen Änderung des Exponenten j werden diese Schritte als „baby steps“ bezeichnet. Anschließend werden die „giant steps“ bestimmt, hierbei wird der Wert des Ausdrucks $(g^{-m})^i$ für alle i mit $0 \leq i < m$ bestimmt und mit den Einträgen der Tabelle verglichen. Ganz analog ist die Bezeichnung „giant steps“ darauf zurückzuführen, dass aufgrund der Einflussgröße m im Exponenten größere Schritte zurückgelegt werden. Wird bei der Bestimmung der „giant steps“ für ein bestimmtes i ein Eintrag gefunden, welcher die Gleichung (1) erfüllt, wird x mit $x = im + j$ ausgegeben. [1, S. 352]

Für eine spätere Implementierung bietet es sich an, sich folgende Umformung zu nutze zu machen: $\alpha(g^{-m})^i = \alpha(g^{-m})^{i-1}g^{-m}$. Der Berechnungsaufwand kann hierdurch etwas reduziert werden, da das Ergebnis der aktuellen Iteration aus dem vorhergehenden Schritt bestimmt werden kann. Das Potenzieren mit i während der Iterationen kann somit entfallen.

2.1.2 Der Algorithmus in Pseudocode

Eingabe: Eine zyklische Gruppe $\mathbb{G}$ der Ordnung n mit einem Generator g und ein Element der Gruppe α.
Ausgabe: Der diskrete Logarithmus $x = log_g(\alpha)$

1. Setze $m := ceil(sqrt(n))$.
2. Für alle $j \in \{0, ..., m-1\}$:
   2.1. Berechne g^j und speichere das Tupel (j, g^j) in einer Tabelle.
3. Setze $t := \alpha$.
4. Für alle $i \in \{0, ..., m-1\}$:
   4.1. Suche in der Tabelle nach einem Paar mit $t = g^j$.
   4.2. Wenn ein solches Paar existiert: Gib $im + j$ aus.
   4.3. Wenn nicht: Setze $t := t * g^{-m}$ und fahre fort.

2.1.3 Beispiel

Als Beispiel wählen wir die prime Restklassengruppe $\mathbb{G} := (\mathbb{Z}/37\mathbb{Z})^\times$. Die vorherrschende Operation ist hier die Multiplikation, welche durch $\times$ gekennzeichnet wird. Durch ausprobieren findet man mit $g = 5$ einen Generator dieser Restklassengruppe; dass $g = 5$ tatsächlich ein erzeugendes Element ist soll im Nachfolgenden kurz gezeigt werden.

Bekanntlich ist ein Element g einer zyklischen Gruppe $\mathbb{G}$ ein Erzeuger dieser, wenn $org(\langle g \rangle) = ord(\mathbb{G})$ mit $ord(\mathbb{G}) := |\mathbb{G}|$ gilt. Durch nachrechnen stellt sich heraus, dass $ord(\langle 5 \rangle) = 36 = ord(\mathbb{G})$ und somit $\langle 5 \rangle = (\mathbb{Z}/37\mathbb{Z})^\times$ gilt. Als Eingabe für den Babystep-Giantstep-Algorithmus setzen wir nun $g = 5$.

Definition 2.1. Die Euler'sche $\varphi(n)$-Funktion gibt für eine natürliche Zahl n an, wie viele zu n teilerfremde natürliche Zahlen existieren, welche nicht größer als n sind:

$$\varphi(n) := \left| \{a \in \mathbb{N} \mid 1 \leq a \leq n \wedge ggT(a, n) = 1\} \right|$$

Nun suchen wir mit Hilfe des Babystep-Giantstep-Algorithmus den diskreten Logarithmus von $\alpha := 4$ zur Basis g, also eine Lösung der Gleichung $4 = 5^x \ mod \ 37$.

- Zunächst bestimmen wird die Ordnung der Gruppe; da 37 eine Primzahl ist, folgt für alle $\beta \in \{1, ..., 36\}$: $ggT(\beta, 37) = 1$. Somit gilt offensichtlich $n := \varphi(37) = 36$.
  Anschließend bestimmen wir m wie folgt: $m := \lceil \sqrt{36} \rceil = 6$.

- Es werden für alle $j \in \{0, ..., m-1\}$ die Tupel (j, g^j) bestimmt und in die nachfolgende Tabelle eingetragen:

j	0	1	2	3	4	5
5^j	1	5	25	14	33	17

- Es gilt $5^{-6} \equiv 5^{36-6} \equiv 27 \; mod \; 37$. Nun berechnen wir für $i \in \{0, ..., m-1\}$ den Wert $4 * 27^i$. Sobald dieser Wert in der zweiten Zeile der vorhergehenden Tabelle gefunden, wurde kann j aus dieser bestimmt werden.

i	0	1	2	3	4	5
$4 * 27^i$	4	34	30	**33**	-	-

  Für $j = 4$ erhalten wir $4 * 27^4 = 33$ dieser Wert findet sich auch in der zweiten Zeile der ersten Tabelle, damit terminiert der Algorithmus und es gilt $i = 3$ sowie $j = 4$, durch Einsetzen erhalten wir schließlich unser Endergebnis $x = 3 * 6 + 4 = 22$. Eine kurze Probe zeigt, dass $5^{22} = 4 \; mod \; 37$ gilt und der Algorithmus somit das korrekte Ergebnis liefert.

2.1.4 Laufzeit

Im Nachfolgenden soll die Laufzeit dieses Algorithmus analysiert werden. Schritt eins und drei werden nicht wiederholt, sie liefern also nur einen konstanten Beitrag zur Gesamtlaufzeit. Die Schleifenbedingung in Schritt zwei ist offenbar nur von m abhängig, somit werden alle Anweisungen innerhalb der Schleife genau m-mal durchlaufen. Im vierten Schritt kann die Schleife vorzeitig verlassen werden, im schlimmsten Fall muss allerdings die Schleife auch genau m-mal durchlaufen werden. Bei Anweisung 4.1 zeigt sich, dass eine iterative Suche innerhalb der zuvor erstellten Tabelle nicht sinnvoll ist, da dies die Gesamtlaufzeit wieder auf $\mathcal{O}(n)$ erhöhen würde. Daher bietet es sich an, zur Speicherung der Wertepaare eine Hashtabelle zu verwenden, wobei die g^j als Schlüssel verwendet werden sollten und die die jeweiligen j als Werte. Die Anweisungen 4.2 und 4.3 sind hingegen wieder elementar, also in $\mathcal{O}(1)$.

Daraus resultiert schließlich eine Gesamtlaufzeit von $\mathcal{O}(m)$ bzw. $\mathcal{O}(\sqrt{n})$, was eine wesentliche Verbesserung der naiven Vorgehensweise zur Bestimmung des diskreten Logarithmus darstellt.

2.1.5 Speicherverbrauch

Der Speicherverbrauch des Babystep-Giantstep-Algorithmus wird im Wesentlichen nur von der Tabelle bestimmt, welche m Einträgen nach Berechnung der Babysteps besitzt. Im weiteren Verlauf des Algorithmus muss, abgesehen von temporären Variablen, kein

weiterer Speicher mehr alloziert werden. Der gesamte Speicherbedarf liegt somit in $\mathcal{O}(m)$ bzw. $\mathcal{O}(\sqrt{n})$. [1, S. 353]

2.2 Index-Calculus-Methode [5] {Fabian Grotz}

Idee:
Wir berechnen den 'großen' Logarithmus, indem wir Logarithmen für 'kleine' Elemente aus Z_p berechnen und damit Rückschlüsse auf den 'großen' Logarithmus ziehen.

2.2.1 Vorberechnung

1. Man bestimmt eine Zahl $B < n$ und die Menge $F(B) = \{p_1, p_2, \cdots, p_B\}$ von Primzahlen, die sogenannte Faktorbasis.
2. Man wähle ein C größer als B, z.B. $B + 10$.
3. Man erhält nun C Kongruenzen der Form $\alpha^{x_j} \equiv {p_1}^{a_{1j}} {p_2}^{a_{2j}} \cdots {p_B}^{a_{Bj}} \bmod p$ für $1 \leq j \leq C$.
4. Dies kann man in ein lineares Gleichungssystem mit C Gleichungen umwandeln mit Gleichungen der Form
   $x_j \equiv a_{1j} \log_\alpha p_1 + \cdots + a_{Bj} \log_\alpha p_B \mod p - 1$ für $1 \leq j \leq C$.
5. Man bestimmt x-Werte, so dass α^x nur Primfaktoren in $F(B)$ hat, und berechnet die Exponenten der Primfaktoren durch Division.
6. Anschließend löst man das lineare Gleichungssystem mit dem Gauss'schen Algorithmus.

- Die Vorberechnung terminiert in $\mathrm{O}(e^{(1+o(1))\sqrt{\ln(p)\ln(\ln(p))}})$.

2.2.2 Bestimmen des diskreten Logarithmus

1. Man wählt zufällig ein s im Bereich $1 \leq s \leq p-2$ und berechnet $y = \beta\alpha^s \bmod p$.
2. Wenn s nur Primfaktoren in $F(B)$ hat, so erhält man
   $\beta\alpha^s \equiv p^{c_1 1} p^{c_2 2} \cdots p^{c_B B} \bmod p$
   ansonsten muss man ein neues s wählen.
3. Jetzt kann man umformen nach
   $\log_\alpha \beta + s \equiv c_1 \log_\alpha p_1 \; c_2 \log_\alpha p_2 \; \cdots \; c_B log_\alpha p_B \mod p - 1$.

- Dieser Algorithmus terminiert in $O(e^{(1/2+o(1))\sqrt{\ln(p)\ln(\ln(p))}})$.

2.2.3 Beispiel

Wir nehmen an, dass $p = 83$ ist. Wir bestimmen nun $B = 7$ und $F(B) = p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7$ als die Faktorbasis aus Primzahlen. Wir wählen außerdem ein $C = 17$.

In folgender Tabelle betrachten wir alles mit mod 82.

$$\begin{aligned}
2^1 &\equiv 2 \\
2^7 &\equiv 45 = 3^2 \cdot 5 \\
2^8 &\equiv 7 \\
2^9 &\equiv 14 = 2 \cdot 7 \\
2^{10} &\equiv 28 = 2^2 \cdot 7 \\
2^{11} &\equiv 56 = 2^3 \cdot 7 \\
2^{12} &\equiv 29 \\
2^{13} &\equiv 58 = 2 \cdot 29 \\
2^{14} &\equiv 33 = 3 \cdot 11 \\
2^{15} &\equiv 66 = 3 \cdot 2 \cdot 11 \\
2^{16} &\equiv 49 = 7^2 \\
2^{17} &\equiv 15 = 3 \cdot 5
\end{aligned}$$

Aus diesem System nehmen wir nur diejenigen Gleichungen heraus, die wir durch unsere vorher bestimmte Faktorbasis darstellen können. In unserem Fall $2^1, 2^7, 2^8, 2^9$ *und* 2^{17}. Wir hätten auch mehr zur Verfügung, aber diese Anzahl reicht bereits.

Die Gleichungen geben uns ein lineares Gleichungssystem das wir lösen. Damit sind

2	3	5	7	
1	0	0	0	1
0	2	1	0	7
0	0	0	1	8
1	0	0	1	9
0	1	1	0	17
0	1	0	0	- 10 = 72
0	0	1	0	34 - 7 = 27

unsere Vorbereitungen abgeschlossen und wir bekommen:

$$\log_2(2) = 1,\ \log_2(3) = 72,\ \log_2(5) = 27,\ \log_2(7) = 8$$

Wenn wir nun $\log_2(31)$ wissen möchten gehen wir wie folgt vor.

$$\begin{aligned} 31^2 &\equiv 48 = 2^4 \cdot 3 \quad |log \\ 2 \cdot \log_2(31) &\equiv 4 \cdot \log_2 2 + \log_2 3 \\ \iff 2 \cdot \log_2(31) &\equiv 1 + 1 + 1 + 1 + 72 = 76 \\ daraus folgt: & \\ \log_2(31) &\equiv 38 \; oder \\ \log_2(31) &\equiv 38 + 41 = 79 \end{aligned}$$

und wie wir aus unserer Annahme leicht nachprüfen können ist $2^{38} \bmod 83 = 31$. Der wert 38 + 41 ist durch das Modulorechnen und die Teilung durch 2 zu überprüfen und mit einzuplanen.

2.2.4 Bemerkung

Von einem Laufzeitstandpunkt aus gesehen ist erst bei großen Primzahlen das Index-Calculus-Verfahren schneller als z.b. der Baby-Step-Giant-Step Algorithmus.

2.3 Fazit {Josef Schmeißer}

Die Laufzeiten der vorgestellten Algorithmen zur Bestimmung des diskreten Logarithmus sind alle von der Gruppenordnung abhängig. Im Gegensatz dazu lässt sich die diskrete Exponentialfunktion direkt berechnen. Da nur elementare Rechenoperationen notwendig sind, kann ein Funktionswert dieser Funktion in konstanter Zeit bestimmt werden.

Allgemein ist im Moment auch kein effizientes Verfahren zur Bestimmung des diskreten Logarithmus bekannt, daher qualifiziert sich die diskrete Exponentialfunktion ebenso wie die Multiplikation großer Primzahlen als Einwegfunktion. Wobei eine sogenannte Einwegfunktion eine Funktion ist, deren Umkehrung zwar existiert, die Bestimmung eines Urbildes ist jedoch mit sehr großem Aufwand verbunden. [2, S. 87]

3 Anwendungen in der Kryptographie

3.1 ElGamal-Verschlüsselungsverfahren {Josef Schmeißer}

Aus der Begebenheit, dass zur Bestimmung des diskreten Logarithmus kein derzeit umzusetzendes effizientes Verfahren bekannt ist, ergeben sich einige Anwendungen in der Kryptographie. Einer dieser Anwendungsfälle ist ein nach ElGamal benanntes Verschlüsselungsverfahren. Hierbei handelt es sich um ein asymmetrisches Verschlüsselungsverfahren.

Hierzu noch eine kurze Erläuterung: Das ElGamal-Verschlüsselungsverfahren nutzt einen öffentlichen und einen geheimen Schlüssel. Der öffentliche Schlüssel dient dem Verschlüsseln von Nachrichten und darf veröffentlicht werden. Der geheime Schlüssel

wird dazu genutzt, empfangene Nachrichten zu entschlüsseln. Der geheime Schlüssel darf natürlich nur dem Empfänger der Nachricht bekannt sein.

Weiterhin ist anzumerken, dass das ElGamal-Verfahren eng mit dem Diffie-Hellman-Schema verwandt ist. Dies ist kein Zufall, denn Taher El Gamal erkannte 1985, dass das Diffie-Hellman-Schlüsselaustauschverfahren zu einem Public-Key-Verschlüsselungsverfahren erweitert werden kann. [1, S. 400] Das Diffie-Hellman-Schlüsselaustauschverfahren soll allerdings hier nicht weiter erläutert werden.

3.1.1 Beschreibung des Verfahrens

Im Nachfolgenden gehen wir davon aus, dass A eine Nachricht an B senden will, wir bezeichnen also den Sender mit A sowie den Empfänger mit B.

Wahl der Parameter

Beide Teilnehmer müssen sich über eine zyklische Gruppe $\mathbb{G} = \langle g \rangle$ der Ordnung p einigen, sowohl $\mathbb{G}$ als auch der Erzeuger g werden öffentlich gemacht. [2, S. 147] Das National Institute of Standards and Technology (oder kurz NIST) hat hierzu eine Auswahl an empfohlenen Parametern veröffentlicht. Generell sollte die Ordnung p der gewählten Gruppe prim sein. [1, S. 404]

Erzeugung der Schlüssel

Der Empfänger muss zunächst eine zufällige natürliche Zahl $b \in \{1, ..., p-1\}$ als geheimen Schlüssel wählen. Anschließend erzeugt der Empfänger B seinen öffentlichen Schlüssel. Er berechnet hierfür das Gruppenelement $\beta = g^b \in \mathbb{G}$ und gibt dieses bekannt.

Verschlüsseln

Möchte nun A eine Nachricht $m \in \mathbb{G}$ an B senden, muss A zunächst eine zufällige natürlich Zahl $a \in \{1, ..., p-1\}$ wählen und daraus das Gruppenelement $\alpha = g^a \in \mathbb{G}$ berechnen. Anschließend bestimmt A den Schlüssel $k = \beta^a \in \mathbb{G}$. A wendet nun einen Verschlüsselungsalgorithmus f unter Verwendung des Schlüssel k auf die Nachricht m an. Den resultierenden Geheimtext $c = f_k(m) \in \mathbb{G}$ sowie das Element α sendet A an B. Im ursprünglichen ElGamal-Schema wurde f definiert als: $f_k(x) = k * x \; mod \; p$ in einer primen Restklassengruppe der Ordnung p. [2, S. 147]

Entschlüsseln

Sobald B eine Nachricht von A empfangen hat, kann B den Schlüssel k aus α und aus b wie folgt bestimmen: $k = \alpha^b \in \mathbb{G}$. Diese Beziehung gilt, da $\alpha^b = g^{ab} \in \mathbb{G}$ in der gewählten Gruppe äquivalent ist zu $\beta^a = g^{ba} \in \mathbb{G}$. Damit kann nun der Geheimtext entschlüsselt werden, indem einfach die Umkehrfunktion f_k^{-1} des Verschlüsselungsalgorithmus angewandt wird.

3.1.2 Beispiel

Wahl der Parameter

Für unser Beispiel wählen wir wie üblich eine zyklische Restklassengruppe $\mathbb{G} := (\mathbb{Z}/p\mathbb{Z})^\times$, mit p prim. Konkret wählen wir die Gruppe $\mathbb{G} = (\mathbb{Z}/47\mathbb{Z})^\times$ und wie schon im Beispiel zum Babystep-Giantstep-Algorithmus stellt sich heraus, dass $g = 5$ ein geeigneter Erzeuger unserer zyklischen Gruppe ist. Anschließend werden die Parameter $p = 47$ und $g = 5$, welche die Gruppe definieren, veröffentlicht.

Erzeugung der Schlüssel

Teilnehmer B wählt $b = 29 \in \{1, ..., p-1\}$ als privaten Schlüssel und berechnet daraus seinen öffentlichen Schlüssel $\beta = g^b = 26 \equiv 5^{29}\ mod\ 47$. Im Anschluss veröffentlicht B den öffentlichen Schlüssel.

Verschlüsseln

In unserem Beispiel möchte Teilnehmer A die Nachricht $m = 42$ an B senden. Dazu wählt Teilnehmer A das zufällige Element $a = 7 \in \{1, ..., p-1\}$ und berechnet daraus das Gruppenelement $\alpha = g^a = 11 \equiv 5^7\ mod\ 47$. Anschließend bestimmt A den Schlüssel $k = \beta^a = 10 \equiv 26^7\ mod\ 47$.

Wie im ursprünglichen ElGamal-Schema wenden wir nun die die Funktion f_k auf m an, um den Geheimtext c zu erhalten: $c = f_{10}(42) = 44 \equiv 10 * 42\ mod\ 47$. Daraufhin übermittelt A den Geheimtext $c = 44$ an B.

Entschlüsseln

Der Empfänger B berechnet zunächst den Schlüssel $k = \alpha^b = 10 \equiv 11^{29}\ mod\ 47$. Mit der Umkehrfunktion $f_{10}^{-1}(x) = 10^{-1} * x\ mod\ 47$ lässt sich nun die ursprüngliche Nachricht m berechnen. Allerdings müssen wir zuerst das multiplikative Inverse von 10 in unserer Gruppe $\mathbb{G}$ bestimmen. Dazu lösen wir mit Hilfe des erweiterten Euklidischen Algorithmus die diphanatische Gleichung $10x + 47y = 1$.

Euklidscher Algorithmus		Rest	
$47 = 4 * 10 + 7$	$\Longrightarrow$	$r_1 = 47 - 4 * 10 = 7$	(1)
$10 = 1 * 7 + 3$	$\Longrightarrow$	$r_2 = 10 - 1 * 7 = 3$	(2)
$7 = 2 * 3 + 1$	$\Longrightarrow$	$r_3 = 7 - 2 * 3 = 1$	(3)
$3 = 3 * 1$	$\Longrightarrow$	$r_4 = 0$	(4)

Damit ist $ggT(47, 10) = 1$ und der Algorithmus ist beendet. Aus den gewonnen Gleichungen können wir nun durch sukzessives rückwärts Einsetzen die ursprüngliche Fragestellung lösen:

Gleichung 3: $1 = 7 - 2 * 3$

Gleichung 2 für 3 einsetzen: $1 = 7 - 2 * (10 - 1 * 7) = -2 * 10 + 3 * 7$

Gleichung 1 für 7 einsetzen: $1 = -2 * 10 + 3 * (47 - 4 * 10) = 3 * 47 - 14 * 10$

Es folgt: $3 * 47 - 14 * 10 = 1$

Eine Lösung dieser Gleichung ist also durch $x = -14$ und $y = 3$ gegeben. Das multiplikative Inverse von 10 mod 47 ist also 10^{-1} mod 47 $= -14$ mod 47 $= 33$ mod 47. Damit ist $f_{10}^{-1}(x) = 33 * x \; mod \; 47$ und wir können schließlich m bestimmen:

$$m = f_{10}^{-1}(44) = 33 * 44 \; mod \; 47 = 42,$$

wie sich zeigt, ist dies — wie zu erwarten war — die richtige Antwort.

3.1.3 Sicherheit

Bevor wir mit dem eigentlichen Sicherheitbeweis beginnen, wollen wir noch einmal kurz das Decisional-Diffie-Hellman-Problem (oder auch kurz DDH) erläutern, da dies direkt in Bezug zum diskreten Logarithmus-Problem steht und für den nachfolgenden Sicherheitbeweis von entscheidender Bedeutung ist.

Das DDH-Problem befasst sich mit der Schwierigkeit, zu entscheiden, ob eine Zahl eine bestimmte Form erfüllt. Die Annahme besagt nun, dass dieses Problem für bestimmte Gruppen schwer ist, es existiert also kein probabilistischer Polynomialzeitalgorithmus (oder auch kurz PPT für probabilistic polynomial time) mit kleiner Fehlerwahrscheinlichkeit, der dieses Problem löst. Auch wenn die sogenannten Diffie-Hellman-Probleme eng mit dem diskreten Logarithmus-Problem verwandt sind, ist nicht bekannt ob diese auch äquivalent sind. [1, S. 320]

Nachfolgenden verwenden wir folgende Notationen: Mit $\mathcal{G}$ bezeichnen wir einen Algorithmus zu Erzeugung von Gruppen. Sei nun $\mathcal{G}$ ein PPT-Algorithmus, welcher einen Sicherheitsparameter 1^n entgegen nimmt und als Ausgabe eine vollständige Beschreibung einer zyklischen Gruppe $\mathbb{G}$, deren Ordnung q und einen Generator g liefert. Das DDH-Problem wird nun wie folgt definiert:

Definition 3.1. Wir sagen, dass ein DDH-Problem schwer in Bezug auf $\mathcal{G}$ ist, wenn für alle PPT-Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion negl existiert, sodass:

$$\left|\Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\right| \leq \mathsf{negl}(n),$$

wobei in beiden Fällen die Wahrscheinlichkeiten über ein Experiment berechnet wird, dass ein Tupel $(\mathbb{G}, q, g)$ als Ausgabe von $\mathcal{G}(1^n)$ bestimmt. Die $x, y, z \in (\mathbb{Z}/q\mathbb{Z})^\times$ werden zufällig gewählt.

Der Ursprung des Parameters 1^n ist vielleicht nicht sofort offensichtlich, deshalb wollen wir an dieser Stelle eine kurze Erläuterung durch folgendes Zitat wiedergeben:

> „**The 1^n hack** Notice the presence of a rather mysterious-looking 1^n as an input parameter to many algorithms [...]. This is a common "hack" found in complexity-theory literature. We want the running time of these algorithms to by polynomial-time in the security parameter n – by passing 1^n as a parameter, we ensure that the input length is of length at least n; and hence the algorithm gets to run for time polynomial in n." [3]

Bevor wir uns wieder dem DDH-Problem zuwenden, wollen wir noch ein kurzes Lemma einführen. Sei $\mathbb{G}$ eine endliche Gruppe und $m \in \mathbb{G}$ ein zufälliges Element. Das Lemma besagt nun, dass die Multiplikation von m mit einem zufälligen Element k der Gruppe, zu gleichmäßig verteilten Elementen k' der Gruppe führt. Somit ist die Verteilung der k' nicht von m abhängig und damit kann aus k' keine Information gewonnen werden, welche Rückschlüsse auf m zulässt. [1, S. 400]

Im Nachfolgenden verwenden wir die Notation $a \leftarrow \mathbb{G}$ für die Wahl eines beliebigen Elements a aus der Gruppe $\mathbb{G}$ sowie für Zuweisungen als Ausgabe von Algorithmen.

Lemma 3.1. Sei $\mathbb{G}$ eine endliche Gruppe und $m \in \mathbb{G}$ willkürlich gewählt. Wird $k \leftarrow \mathbb{G}$ zufällig gewählt und ist $k' := k * m$, dann erhält man die selbe Verteilung für k', als wenn $k' \leftarrow \mathbb{G}$ gewählt wird. Anders gesagt, gilt für alle $a \in \mathbb{G}$:

$$\Pr[k * m = a] = 1/|\mathbb{G}|,$$

wobei die die Wahrscheinlichkeit durch eine zufällige Wahl von $k \in \mathbb{G}$ bestimmt wird.

Beweis. Sei $a \in \mathbb{G}$ beliebig. Dann gilt:

$$\Pr[k * m = a] = \Pr[k = a * m^{-1}].$$

Da k gleichmäßig per Zufall aus $\mathbb{G}$ gewählt wurde, ist die Wahrscheinlichkeit, dass k gleich dem fixierten Element $m^{-1} * a$ ist, genau $1/|\mathbb{G}|$.

□

Dieses soeben bewiesene Lemma lässt sich dazu nutzen, ein perfekt sicheres Verschlüsselungsverfahren mit $\mathbb{G}$ als Nachrichtenraum zu konstruieren. Sender und Empfänger teilen sich den geheimen Schlüssel $k \leftarrow \mathbb{G}$. Eine Nachricht $m \in \mathbb{G}$ kann nun einfach mit $c := k * m$ verschlüsselt und mit $m := c/k$ entschlüsselt werden. [1, S. 400] Ein solches Verfahren wäre einfach eine Umsetzung des One-Time-Pad-Schemas.

Für den anschließenden Sicherheitbeweis wollen wir nun noch eine formale Beschreibung des ElGamal-Verschlüsselungsverfahrens angeben, welches aus den drei Algorithmen Gen, Enc und Dec besteht. In Zukunft bezeichnen wir ein Verschlüsselungsverfahren bestehend aus diesen drei Algorithmen mit Π.

Formale Beschreibung des Verfahrens

- Gen: Die Eingabe des Schlüsselerzeugungsalgorithmus ist ein Sicherheitsparameter 1^n. Damit wird der Gruppenerzeugungsalgorithmus $\mathcal{G}(1^n)$ ausgeführt, um $(\mathbb{G}, q, g)$ zu erhalten. Wähle ein zufälliges $x \in (\mathbb{Z}/q\mathbb{Z})^\times$ und berechne $h := g^x$. Der öffentliche Schlüssel ist gegeben durch $\langle \mathbb{G}, q, g, h \rangle$ und der private Schlüssel mit $\langle \mathbb{G}, q, g, x \rangle$. Der Nachrichtenraum ist $\mathbb{G}$.

- Enc: Der Verschlüsselungsalgorithmus nimmt eine Nachricht $m \in \mathbb{G}$ und den öffentlichen Schlüssel $pk = \langle \mathbb{G}, q, g, x \rangle$ entgegen. Es wird ein zufälliges Element $y \in (\mathbb{Z}/q\mathbb{Z})^\times$ gewählt und als Ausgabe folgender Geheimtext erzeugt:

$$\langle g^y, h^y * m \rangle.$$

- Dec: Der Entschlüsselungsalgorithmus erhält als Eingabe den privaten Schlüssel $sk = \langle \mathbb{G}, q, g, x \rangle$ und den Geheimtext $\langle c_1, c_2 \rangle$. Ausgegeben wird die entschlüsselte Nachricht $\hat{m} \in \mathbb{G}$:

$$\hat{m} := c_2 / c_1^x.$$

Das Public-Key-Experiment, auf das wir uns nachfolgend beziehen, kann für ein gegebenes Verschlüsselungsverfahren $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ und einen Angreifer $\mathcal{A}$, welcher einen Abhörversuch unternimmt, wie folgt beschrieben werden:

Das Ununterscheidbarkeitsexperiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$: [1, S. 379]

- $\mathsf{Gen}(1^n)$ wird ausgeführt, um ein Schlüsselpaar (pk, sk) zu erhalten.

- Der Angreifer $\mathcal{A}$ erhält den öffentlichen Schlüssel pk. Dieser generiert daraufhin zwei Nachrichten gleicher Länge: m_0 und m_1.

- Es wird ein zufälliges Bit $b \in \{0, 1\}$ gewählt, woraufhin der Geheimtext

$$c \leftarrow \mathsf{Enc}_{pk}(m_b)$$

  bestimmt wird und an $\mathcal{A}$ übergeben wird.

- $\mathcal{A}$ gibt ein Bit b' aus. Das Experiment gibt nun 1 zurück wenn $b' = b$ ansonsten 0. Wobei der Angreifer $\mathcal{A}$ Erfolg hatte, wenn $b' = b$ gilt.

Definition 3.2. Das Verschlüsselungsverfahren $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ besitzt nicht zu unterscheidende Geheimtexte aus Sicht eines Abhörers, wenn für alle PPT-Angriffsalgorithmen $\mathcal{A}$ eine vernachlässigbare Funktion negl existiert, für die gilt: [1, S. 379]

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Nun haben wir alle Vorbereitungen getroffen und können nachfolgend die Sicherheit dieses Verfahrens in Bezug auf das DDH-Problem beweisen.

Satz 3.2. Wenn das DDH-Problem schwer in Bezug auf $\mathcal{G}$ ist, dann generiert das ElGamal-Verschlüsselungsverfahren nicht zu unterscheidende Geheimtexte unter Angriffen mit ausgewählten Klartexten. Anders gesagt: das Verfahren garantiert die Ununterscheidbarkeit von Geheimtexten.

Beweis. Sei Π das ElGamal-Verschlüsselungsverfahren. Wir zeigen nachfolgend, dass Π nicht zu unterscheidende Geheimtexte unter Anwesenheit eines Beobachters generiert.

Sei $\mathcal{A}$ ein PPT-Algorithmus, welcher die Rolle des Angreifers übernimmt. Nun wollen wir zeigen, dass eine zu vernachlässigende Funktion negl existiert, sodass gilt:

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Nun führen wir das modifizierte „Verschlüsselungsverfahren“ $\tilde{\Pi}$ ein, dies besitzt die Eigenschaft, dass Gen identisch zur Beschreibung in Π ist, die Verschlüsselung einer Nachricht m mit dem öffentlichen Schlüssel $\langle \mathbb{G}, q, g, h \rangle$ wird allerdings mit zufällig gewählten $y, z \in (\mathbb{Z}/q\mathbb{Z})^\times$ durchgeführt. Der ausgegebene Geheimtext hat die Form:

$$\langle g^y, g^z * m \rangle.$$

Es zeigt sich, dass $\tilde{\Pi}$ kein Verschlüsselungsverfahren ist, da für einen Empfänger keine Möglichkeit besteht, den empfangen Geheimtext zu entschlüsseln. Das Experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ ist allerdings dennoch wohldefiniert, da das Experiment lediglich von der Erzeugung der Schlüssel und dem Entschlüsselungsalgorithmus abhängig ist.

Aus Lemma 3.1 folgt, dass die zweite Komponente des Geheimtextes des Verfahrens $\tilde{\Pi}$ ein gleichmäßig zufällig verteiltes Element der Gruppe $\mathbb{G}$ ist. Weiterhin besteht keine Abhängigkeit zur Nachricht m, welche zu verschlüsseln ist, da aus Lemma 3.1 auch folgt, dass g^z ein zufälliges Element von $\mathbb{G}$ ist, wenn z aus $(\mathbb{Z}/q\mathbb{Z})^\times$ zufällig gewählt wurde. Die erste Komponente des Geheimtextes ist offensichtlich nicht von m abhängig, es lassen sich somit auch keine Informationen über m aus dieser gewinnen. Zusammengefasst bedeutet dies, dass der gesamte Geheimtext keinerlei Informationen über m enthält. Daraus folgt folgender Zusammenhang:

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\tilde{\Pi}}(n) = 1] \leq \frac{1}{2}.$$

Nun wollen wir einen PPT-Algorithmus D betrachten, welcher das gegebene DDH-Problem relativ zu $\mathcal{G}$ zu lösen versucht. Der Algorithmus D erhält das Tupel

$$(\mathbb{G}, q, g, h_1, h_2, h_3),$$

dabei gilt: $h_1 = g^x$, $h_2 = g^y$ und h_3 ist entweder g^{xy} oder g^z, wenn x, y und z zufällig gewählt wurden. Der Algorithmus D sollte nun herausfinden, welcher dieser beiden Fälle zutrifft.

Der Algorithmus D

Eingabe: Das Tupel $(\mathbb{G}, q, g, h_1, h_2, h_3)$
Ausgabe: Ein einzelnes Bit

- Setze $pk = \langle \mathbb{G}, q, g, h_1 \rangle$ und führe $\mathcal{A}(pk)$ aus, um die zwei Nachrichten $m_0, m_1 \in \mathbb{G}$ zu erhalten.
- Wähle ein zufälliges Bit b und setze $c_1 := h_2$ sowie $c_2 := h_3 * m_b$.
- Übergebe den verschlüsselten Text $\langle c_1, c_2 \rangle$ an $\mathcal{A}$ und erhalte das Bit b' als Ausgabe. Wenn $b' = b$ gilt, gebe 1 aus, andernfalls 0.

Wenn wir nun das Verhalten von D analysieren wollen, müssen wir im Nachfolgenden zwei Fälle unterscheiden:

Fall 1: Zunächst wird $\mathcal{G}(1^n)$ ausgeführt um das Tupel $(\mathbb{G}, q, g)$ zu generieren. Anschließend werden zufällig $x, y, z \in (\mathbb{Z}/q\mathbb{Z})^\times$ gewählt und es wird $h_1 := g^x$, $h_2 := g^y$ und $h_3 := g^z$ gesetzt. Der Algorithmus D führt $\mathcal{A}$ mit einem öffentlichen Schlüssel aus, der wie folgt konstruiert wird:

$$pk = \langle \mathbb{G}, q, g, g^x \rangle$$

und einem Geheimtext, der wie folgt konstruiert wird:

$$\langle c_1, c_2 \rangle = \langle g^y, g^z * m_b \rangle.$$

Die Sicht von $\mathcal{A}$, ausgeführt als Unterroutine von D, ergibt eine Verteilung, welche identisch zur Verteilung im Experiment mit $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\tilde{\Pi}}(n)$ ist. Da D nur eine 1 zurück gibt, wenn die Ausgabe b' von $\mathcal{A}$ gleich zu b ist, gilt somit:

$$\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\tilde{\Pi}}(n) = 1] = \frac{1}{2}.$$

Fall 2: Zunächst wird $\mathcal{G}(1^n)$ ausgeführt um das Tupel $(\mathbb{G}, q, g)$ zu generieren. Anschließend werden zufällig $x, y \in (\mathbb{Z}/q\mathbb{Z})^\times$ gewählt und es wird $h_1 := g^x$, $h_2 := g^y$ und $h_3 := g^{xy}$ gesetzt. Der Algorithmus D führt $\mathcal{A}$ mit einem öffentlichen Schlüssel aus, der wie folgt konstruiert wird:

$$pk = \langle \mathbb{G}, q, g, g^x \rangle$$

und einem Geheimtext, der wie folgt konstruiert wird:

$$\langle c_1, c_2 \rangle = \langle g^y, g^{xy} * m_b \rangle = \langle g^y, (g^x)^y * m_b \rangle.$$

Die Sicht von $\mathcal{A}$, ausgeführt als Unterroutine von D, ergibt eine Verteilung, welche identisch zur Verteilung im Experiment mit $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ ist. Da D nur eine 1 zurück gibt, wenn die Ausgabe b' von $\mathcal{A}$ gleich zu b ist, gilt somit:

$$\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}.$$

Unter der Annahme, dass das DDH-Problem schwer in Bezug auf $\mathcal{G}$ ist, existiert eine zu vernachlässigende Funktion negl, sodass gilt:

$$\begin{aligned} \mathsf{negl}(n) &\geq \left| \Pr[D(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[D(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] \right| \\ &= \left| \frac{1}{2} - \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \right|. \end{aligned}$$

Dies impliziert $\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$, was zu zeigen war.

□

3.1.4 Praxis

Das ElGamal-Verschlüsselungsverfahren wird auch in der Praxis eingesetzt, so wird es beispielsweise von GnuPG angeboten. Für einen gewissen Zeitraum war das ElGamal-Verfahren sogar als Voreinstellung in GnuPG gesetzt.

Das Verfahren hat allerdings auch Nachteile, wie beispielsweise der Nachrichtenraum, welcher durch die gewählte Gruppe $\mathbb{G}$ bestimmt wird. Für manche Gruppen ist es jedoch möglich, eine umkehrbare Kodierung von Gruppenelementen zu Bitstrings zu definieren und dieses Problem damit zu umgehen. [1, S. 404]

Weiterhin ist das Verfahren sehr rechenintensiv, da häufig starker Pseudozufallszahlen generiert werden müssen. Es ist daher auch nicht besonders zur Verschlüsselung großer Datenmengen geeignet. Hier bietet es sich an, das ElGamal-Verfahren nur zur Verschlüsselung eines Schlüssels zu verwenden, welcher wiederum für ein symmetrisches Verfahren genutzt wird. Ein solches Konstrukt wird als hybrides Verschlüsselungverfahren bezeichnet. [4]

3.2 Massey-Omura-Schema [6] {Fabian Grotz}

Der Grundgedanke an die Anforderungen dieses Sicheren Übertragungsschemas war, dass beide Parteien weder den privaten noch den öffentlichen Schlüssel der anderen Partei besitzen oder auf diesen Schlussfolgern können. Das Massey-Omura-Schema wurde hierzu im Jahre 1983 von den Kryptologen James Massey und Jim Omura entwickelt und basiert auf der Schwierigkeit, den diskreten Logarithmus zu lösen.

3.2.1 intuitive Beschreibung

Um im Vorhinein die Wirkungsweise zu verdeutlichen ohne die dahinter liegende Mathematik anzutasten, beschreiben wir die Methode mithilfe einer Metapher.

- Alice und Bob wollen Nachrichten austauschen. Zu erst möchte Alice an Bob eine Nachricht schicken und legt diese Nachricht in eine Kiste, verschließt sie mit ihrem Schloss und sendet die Kiste an Bob.
- Bob bekommt die Kiste, kann das Schloss von Alice nicht entfernen, da er den Schlüssel hierzu nicht hat. Nun hängt Bob auch sein Schloss an die Truhe und schickt die Truhe wieder zurück an Alice.
- Alice entfernt ihr eigenes Schloss, da sie ursprünglich den Schlüssel hierzu besitzt. Die Kiste ist immer noch verschlossen, da Bobs Schloss noch an der Kiste hängt. Alice schickt nun die Kiste ein letztes mal an Bob.

- Bob bekommt die Kiste mit nur seinem Schloss das er durch seinen eigenen Schlüssel auch wieder entfernen kann. Die Kiste hat nun kein Schloss mehr und Bob kann die Nachricht von Alice lesen die sich in der Kiste befindet.

3.2.2 Voraussetzung

Die Voraussetzung an das Verfahren ist, dass beide über eine gemeinsame und genügend große Primzahl p bescheid wissen. Diese Primzahl kann öffentlich bekannt sein.

Alice und Bob wählen nun jeweils ein Paar von Exponenten d und e mit

$$ed \equiv 1 \pmod{p-1}$$

, das bedeutet also dass

$$a^{de} \equiv a \pmod{p-1}$$

für alle ganzen Zahlen $a \in Z$ gilt. Diese Behauptung wurde durch den kleinen Satz von Fermet bereits bestätigt. Diese Voraussetzung ist dann später hilfreich um mit dem Schlüsselpaar die eigene Verschlüsselung wieder aufzuheben. Für die Berechnung von e muss zuerst ein e mit

$$e < p-1 \text{ und } ggT(e, p-1) = 1$$

gefunden werden. d ist dann das multiplikativ Inverse von $e \bmod p-1$.
Beide Parteien Alice und Bob halten ihre Zahlen geheim.

3.2.3 Ablauf

Alice berechnet aus ihrer ursprünglichen Nachricht $m < p$ die Nachricht x_A die sie an Bob sendet

$$x_{A1} = m^{e_A} \bmod p$$

Bob berechnet aus dieser Nachricht seine eigene Rückantwort

$$\begin{aligned} x_B &= (x_A)^{e_B} \bmod p \\ \Leftrightarrow x_B &= (m^{e_A})^{e_B} \bmod p. \end{aligned}$$

Alice erzeugt ausgehend der Rückantwort von Bob und mithilfe ihres Schlüssels nun die nur von Bob verschlüsselte Nachricht.

$$\begin{aligned} x_{A2} &= {x_B}^{d_A} \bmod p \\ \Leftrightarrow x_{A2} &= ((m^{e_A})^{e_B})^{d_A} \bmod p \\ \Leftrightarrow x_{A2} &= ((m^{e_A})^{d_A})^{e_B} \bmod p \\ \Leftrightarrow x_{A2} &= m^{e_B} \bmod p \end{aligned}$$

In diesem Schritt hat Alice also all ihre Verschlüsselung der Ursprungsnachricht wieder aufgehoben und die Nachricht ist nur noch durch die Verschlüsselung von Bob verschlüsselt.
Bob entschlüsselt die Nachricht nun mit

$$x_{A2}{}^{d_B} = (m^{e_B})^{d_B} = m \bmod p.$$

Bob hat damit nun die Ursprungsnachricht von Alice bekommen. Außer Alice und Bob konnte ohne das Wissen der Schlüssel niemand auf die Ursprungsnachricht m während der Datenübertragung zugreifen.

3.2.4 Sicherheitsaspekte

Das Massey-Omura-Schema ist gegen einen Lauschangriff sicher, solange das Problem des Diskreten Logarithmus nicht effizient lösbar ist. Wenn dieser Fall eintreten sollte, ist dieses Verfahren nicht mehr sicher, da aus den einzelnen übertragenen Nachrichten Rückschlüsse auf die Schlüssel gezogen werden könnten.
Einen Wirkungsvollen Schutz gegen einen Man-in-the-Middle Angriff bietet dieses Schema jedoch nicht, da dieses Verfahren nur gegen Lauschangriffe ausgelegt ist.

4 Ausblick {Fabian Grotz}

Wie wir mehrmals darauf hingewiesen haben, ist der Diskrete Logarithmus mit derzeitigen Rechenleistungen und Berechnungsverfahren nicht in sinnvoller Zeit berechenbar. In ferner Zukunft könnte sich dies jedoch mit der Einführung schneller und optimierter Quantencomputer ändern. In der damit neu entstehenden Quanteninformatik wird dadurch ein Algorithmus berechenbar, der die Quantentheorie benutzt um das diskrete Logarithmusproblem essentiell schneller berechnen kann als jeder bisherige Algorithmus. Dieser Algorithmus wird Shor-Algorithmus genannt. Leider (oder besser gesagt, gut für uns) ist dem Algorithmus derzeit noch eine Technische Hürde auferlegt, da zur Berechnung einer Zahl n man einen Quantencomputer mit $log(n)$ Qubits benötigt. Neueste Forschungen haben die Qubit-zahl zwar bereits über 1000 gehoben, dies ist jedoch bisher reiner Gegenstand der Forschung.
Derzeit werden alle Quantencomputer nur für einen spezifischen Zweck gebaut, laufen nur unter Laborbedingungen und sind noch nicht universell programmierbar. Dies wird sich jedoch in Zukunft ändern und spätestens dann muss ein neuer Algorithmus bzw. neue Verfahren gefunden werden, wie trotz oder auch mit Quantencomputern Übertragungen verschlüsselt werden können.

5 Literaturverzeichnis

[1] Jonathan Katz und Yehuda Lindell. Introduction to Modern Cryptography. CRC Press, second edition, 2015.

[2] Albrecht Beutelspacher. Kryptologie – Eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen. Springer Spektrum, 10. Auflage, 2015.

[3] Arshed Nabeel. Principles of Modern Cryptography – Provable Security. http://drona.csa.iisc.ernet.in/~arpita/Cryptography15/Scribe1.pdf, 2015. Zugriff: 19.04.2015 19:30.

[4] Allison Bishop. Lecture 3: El Gamal Encryption. https://www.cs.utexas.edu/~rashid/395Tcrypt/2_1.pdf, 2009. Zugriff: 14.04.2015 11:00.

[5] In Anlehnung an Carsten Baum http://www.cs.uni-potsdam.de/ti/lehre/09-Kryptographie/slides/Baum_PublicKey1.pdf

[6] In Anlehnung an Klaus Pommerening, https://www.staff.uni-mainz.de/pommeren/Kryptologie/Asymmetrisch/4_DiskrLog/nokey.pdf